A CONSTRAINT-BASED FRAMEWORK TO MODEL HARMONY FOR ALGORITHMIC COMPOSITION

Torsten Anders
University of Bedfordshire
Torsten.Anders@beds.ac.uk

ABSTRACT

Music constraint systems provide a rule-based approach to composition. Existing systems allow users to constrain the harmony, but the constrainable harmonic information is restricted to pitches and intervals between pitches. More abstract analytical information such as chord or scale types, their root, scale degrees, enharmonic note representations, whether a note is the third or fifth of a chord and so forth are not supported. However, such information is important for modelling various music theories.

This research proposes a framework for modelling harmony at a high level of abstraction. It explicitly represents various analytical information to allow for complex theories of harmony. It is designed for efficient propagation-based constraint solvers. The framework supports the common 12-tone equal temperament, and arbitrary other equal temperaments. Users develop harmony models by applying user-defined constraints to its music representation.

Three examples demonstrate the expressive power of the framework: (1) an automatic melody harmonisation with a simple harmony model; (2) a more complex model implementing large parts of Schoenberg's tonal theory of harmony; and (3) a composition in extended tonality. Schoenberg's comprehensive theory of harmony has not been computationally modelled before, neither with constraints programming nor in any other way.

1. INTRODUCTION

In the field of algorithmic composition, harmony is a challenging area to address. Theories of harmony can be rather complex, as the mere size of standard harmony textbooks indicates. Also, different theories vary considerably depending on the musical style they address such as classical music [1], Jazz [2], contemporary classical music in extended tonality [3], or microtonal music [4].

As there is no agreement on a single theory of harmony, a flexible algorithmic composition environment should allow users to define their own theory. The present research provides a framework by which users can model their own theory of harmony, and then let the system generate music that follows it.

The framework provides building blocks common to many theories and that way simplifies the definition of custom theories from scratch. The proposed framework provides flexible representations of harmonic concepts (e.g., chords, scales, notes, as well as their parameters like a note's pitch or a chord root), which allow users to define their own harmony models declaratively at a high-level of abstraction with modular rules implemented by constraints that restrict the relations between these parameters.

Users can freely declare chord and scale types (e.g., major and minor triads and scales) by specifying pitch class intervals among chord or scale tones and their root. A number of different pitch representations are supported including pitch numbers, pitch classes, enharmonic note representations, scale degrees of notes and chords, and specific chord tones such as the fifth or the third of a triad. All these representations are freely constrainable. The framework supports the common 12-tone equal division of the octave (12-EDO) and arbitrary other equal divisions.

These models can generate harmonic progressions. If the harmonic rules are complemented by rules controlling the melody, counterpoint and so forth, then they can also be used more generally to generate music that follows given or generated progressions.

The presented framework is implemented in the music constraint system Strasheela [1] on top of the Oz programming language [5].

1.1 Plan of Paper

The rest of this paper is organised as follows. Section 2 puts this research into the context of previous work. Application examples of the proposed framework are shown in Sec. 3, while Sec. 4 presents formal details of the framework. Section 5 shows how to develop harmony models with this framework. The paper closes with a discussion that points out limitations of the proposal (Sec. 6).

2. BACKGROUND

Several algorithmic composition systems offer a rich representation of pitch-related and harmonic information, e.g., Impromtu, Opusmodus, the pattern language of SuperCollider, or music analysis systems like Humdrum and music21. However, such systems process this information procedurally, which makes it difficult to model the complex interdependencies found in harmony.

[1] http://strasheela.sourceforge.net/

The declarative approach of constraint programming [6] is more suitable for that. In this programming paradigm, users define *constraint satisfaction problems* (CSP) in which *constraints* restrict relations between *decision variables*. [2] Variables are unknowns, and initially have a *domain* of multiple possible values. A *solver* then searches for one or more *solutions* that reduces the domain of each variable to one value that is consistent with all its constraints.

Several surveys, partly by this author, give an overview of how constraint programming has been used for modelling music theory and composition. Pachet and Roy [7] focus on constraint-based harmonisation, specifically four-part harmonisation of a given melody. An introduction to the field of modelling composition and music theory with constraint programming in general is the detailed review [8]. A further review [9] presents how six composers used constraint programming to realise specific compositions.

A particular extensive constraint-based harmony system is CHORAL by Kemal Ebcioğlu [10], which generates four-part harmonisations for given choral melodies that resemble the style of Johann Sebastian Bach.

Several existing systems allow users to define their own harmonic CSPs, and each of these systems has its specific advantages. Situation [11] provides a family of mini-languages for specifying how the arguments to certain predefined harmonic constraints change across chords. ScorePMC, as subsystem of PWConstraints [12, Chap. 5] is well suited for complex polyphonic CSPs where the rhythm is fixed. PWMC [13] and its successor Cluster Engine allow for complex polyphonic CSPs where the rhythm is constrained as well.

While all these systems allow users to define their own harmonic CSPs, they restrict the constrainable harmonic information to pitches and intervals between pitches. For example, users can constrain harmonic and melodic intervals between note pitches, or represent an underlying harmony by chords consisting of concrete notes. However, the music representations of these systems are not extendable, and more abstract analytical information such as chord and scale types or roots, scale degrees, enharmonic note representations etc. are not supported. Such analytical information is important for various music theories. Some information could be deduced from a pitch-based representation (e.g., pitch classes), but other information is difficult to deduce (e.g., the chord type or root) and complex harmonic CSPs are more difficult to define and solve this way.

An expressive harmonic constraint system that supports such analytical information is the combination of the music representation MusES [14] and the constraint solver BackTalk [15]. This combination was used, e.g., for automatic harmonisation. BackTalk supports variable domains of arbitrary SmallTalk collections. The search algorithm first filters all variable domains with a general consistency-checking algorithm, and then searches for a solution of this reduced problem with a backjumping algorithm.

Many modern constraint solvers use constraint propagation for efficiency [16, Chap. 4]. Their variable domains are restricted to specific types (e.g., Boolean, integer and sets of integers), and highly optimised domain-specific propagation algorithms filter variable domains before every step in the search process, which typically greatly reduces the search space. The design of MusES cannot be ported to propagation-based constraint solvers, because it is not limited to the variable types supported by such solvers. MusES has originally not been designed with constraint programming in mind, and the variable domains of CSPs defined with MusES consist of complex SmallTalk objects.

This research proposes the first harmony framework that supports various analytical information to allow for modelling complex theories of harmony at a high level of abstraction, and whose design is at the same time suitable for propagation-based constraint solvers. Using constraint propagation allows, e.g., to quickly solve CSPs with large domain sizes and that way allows for microtonal harmony. While the framework was implemented in Strasheela on top of Oz, it could also be implemented with any other constraint system that supports the following features found in various propagation-based constraint systems (e.g., Gecode, Choco, JaCoP, MiniZinc): the variable domains Boolean, integer, and set of integers; integer and set constraints including reified constraints (meta-constraints, which also constrain whether their stated relationship holds or not); and the element constraint. [3]

This framework is largely comparable in terms of its flexibility with the combination of MusES and BackTalk, although it has a different stylistic focus. MusES was designed for jazz, while this research focuses on classical tonal music and contemporary music in an extended tonality including microtonal music.

3. APPLICATIONS

Before presenting formal details of the proposed framework, some examples showing the framework in action will further motivate this research. Please remember that all rules discussed for these examples are only a demonstration of the capabilities of the proposed framework, and all these rules can of course be changed.

3.1 Automatic Melody Harmonisation

The first example creates a harmonisation for a given melody. It is comparatively simple, and is therefore discussed in more detail.

While the proposed framework was originally developed for music composition, it can also be used for analysis, because it only controls relations between concepts like notes and chords. This example demonstrates an intermediate case. It performs an automatic harmonic analysis of a given folk tune, but additional compositional rules are applied to the resulting harmonies. Voicing is irrelevant in this example; only the chord symbols are searched for.

The harmonic rhythm is slower than the melody, as common is classical, folk and popular music. By contrast, most automatic harmonisation examples in the literature are choral-like with one chord per note.

[2] In the rest of the paper we just use the term *variable* for brevity.

[3] For details on the element constraint see `https://sofdem.github.io/gccat/gccat/Celement.html`, and also section 4.3.

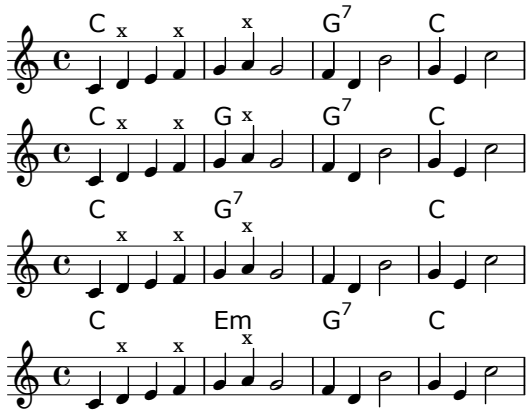

Figure 1. For the given melody this simple harmonisation model has four solutions

For simplicity, this example defines relatively rigid basic conditions. Only major, minor, and dominant seventh chords are permitted, and all chords must be diatonic in C major. The harmonic rhythm is fixed, and all chords share the same duration (e.g., a whole bar), but chord repetitions are permitted.

The example distinguishes between harmonic and nonharmonic tones, but for simplicity only a few cases of nonharmonic tones are permitted (passing and neighbour tones). All other melody notes must be chord pitches.

The example further borrows a few harmonic rules from Schoenberg [1] in order to ensure musically reasonable solutions. The example assumes that the given melody starts and ends with the tonic – these chords are constrained to be equal. A seventh chord must be resolved by a 'fourth upwards the fundament' (e.g., $V^7 \rightarrow I$), the simplest resolution form for seventh chords. All chords share at least one common pitch class with their predecessor (harmonic band, a simpler form of Schoenberg's directions for producing favourable chord progressions).

Figure 1 shows all solutions for the first phrase of the German folksong "Horch was kommt von draussen 'rein" that fulfil the given rules. An x on top of a note denotes a nonharmonic pitch.

Because of the relative simplicity of this example, it works only well for some melodies and less good for others. The harmonic rhythm of the melody must fit the harmonic rhythm specified for the example (at least chords can last longer, as repetitions are permitted). This is easily addressed by turning the currently fixed chord durations into variables, but doing so increases the size of the search space. Further, the nonharmonic pitches of the melody must fit the cases defined (passing and neighbour tones). An extension could define further cases, like suspensions and anticipations. Also, the melody currently cannot modulate. This can be solved by additionally modelling scales, and applying modulation constraints between chords and scales (as shown in the next example).

3.2 Modelling Schoenberg's Theory of Harmony

The next example implements large parts of Schoenberg's tonal theory of harmony [1] – a particular comprehensive theory – and that way demonstrates that the framework is capable of modelling complex conventional theories. To the knowledge of the author, Schoenberg's theory of harmony has not been computationally modelled before, neither with constraints programming nor in any other way. Also, this example implements modulation, which has rarely been done with constraint programming before. Among the many proposed systems, Ebcioğlu's CHORAL [10] is the only system the author is aware of that supports modulation. As the current example implements a lot of harmonic knowledge, it can only be summarised here.

The example modulates from C major to G major, but features an extended pitch set that allows for non-diatonic tones. Figure 2 shows a solution.

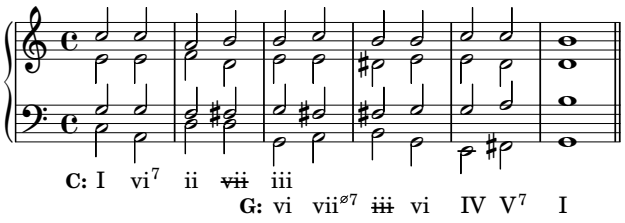

Figure 2. A solution of the Schoenberg harmony model modulating from C to G major; Roman numerals added manually for clarity (altered chords are crossed out)

The example applies Schoenberg's guidelines on chord root progressions designed to help obtaining better progressions. Schoenberg distinguishes between three kinds of root progressions. In *strong* or *ascending* progressions the chord root progresses by a fourth up (harmonically the same as a fifth down) or a third down. For example, in Fig. 2 chord 1 progresses to chord 2 by a third down (I $\rightarrow$ vi), while chord 2 progresses to chord 3 by a fourth up / fifth down (vi $\rightarrow$ ii). *Descending* root progressions form the second type. They are the inversion of strong progressions: a fourth down (fifth up) or a third up. These are not allowed in this example. Finally, in *superstrong* progressions the root moves a second up or down, as the chords in the penultimate bar do (IV $\rightarrow$ V). These different progressions are explained in more detail and formalised in [17].

The chords are related to underlying scales, which change during the modulation. The first five chords relate to the C major scale, and the rest to G major. However, the example also allows for non-diatonic tones. Schoenberg introduces these as accidentals from church modes, namely the raised 1st, 2nd, 4th, 5th and the flattened 7th degree. These are always resolved stepwise in the direction of their alteration (e.g., raised degrees go up). In order to support the intended direction of the modulation (C to G), only raised degrees are allowed in this example. In the solution above, e.g., chord 4 contains a raised 4th degree of C major (F♯) that is chromatically introduced and resolves upwards by a step. Chord 7 contains a raised 5th degree of G major (D♯).

A modulation constraint requires that chord 5 is a *neutral chord*, i.e., a chord shared by both keys. In the solution above this chord is iii in C and vi in G major. The modulation constraint further requires that the neutral chord is followed by the *modulatory chord*, which is only part of

the target key (vii$^{\varnothing 7}$ in the solution). For clarity, the modulatory chord must progress by a strong root progression a fourth upwards (into ~~iii~~ above).

The example also applies voice leading rules. Open and hidden parallel fifths and octaves are not permitted. Also, the upper three voices are restricted to small melodic intervals and small harmonic intervals between voices (larger harmonic intervals are allowed between tenor and bass).

Root positions and first inversions can occur freely. For example, chord 4 is a first inversion in Fig. 2. The number of first inversions has not been constrained and it is rather high in the shown solution (6 chords out of 11). More generally, statistical properties such as the likelihood of these chords are difficult to control by constraint programming (e.g., their overall number can be restricted, but they then may be bunched early or late in the search process).

It should be noted that the pitch resolution of this example is actually 31-tones per octave (31-EDO). This temperament is virtually the same as quarter-comma meantone, a common tuning in the 16th and 17th century. It can be notated with standard accidentals (♯, ♭, 𝄪 ...). This temperament has been used here, because it simplifies the notation by distinguishing between enharmonically equivalent pitches (e.g., E♭ and D♯ are different pitch classes in this temperament). More generally, the use of this temperament demonstrates that the proposed framework supports microtonal music [18].

3.3 A Compositional Application in Extended Tonality

The last example discusses the 7 minute composition *Pfeifenspiel* by the author, which was composed for the two organs of the Kunst-Station St. Peter in Köln (premiered at the Computing Music VIII series in 2012). An excerpt from the piece is shown in Fig. 3 on the following page.

The music is tonal in the extended sense of Tymoczko [19]: melodic intervals tend to be small; the dissonance degree of the harmony is rather consistent; relatively consonant chords are used in moments of musical stability; sections of the piece are limited to certain scales; and for specific sections one tone is particularly important (root).

However, the piece is clearly non-diatonic. Suitable scales where found by first searching with an ad-hoc constraint program through about 200 scale and 50 chord types for scales that contain many chords with a similar dissonance degree (measured with an experimental algorithm). Solution scales were further evaluated manually by considering all chords that can be built on each scale degree, and by judging the melodic quality of scales. In the end, three scales that are all somewhat similar to the whole tone scale where selected: Takemitsu's *Tree Line* mode 2, Messiaen's mode 3, and Messiaen's mode 6. Two of these scales are shown in Fig. 3 in the analysis in the lowest stave (e.g., Takemitsu's *Tree Line* mode 2 on D in measures 6–7).

Based on these scales, a global harmonic and formal plan was composed by hand, but concrete harmonic progressions were generated algorithmically with custom harmony models for different sections. Also, contrapuntal sections rendering the harmony were algorithmically generated (and slightly manually revised), while some other sections were composed manually (e.g., in Fig. 3 the septuplets in the Great division, and the triplets in the pedal were composed manually).

Some example constraints are outlined. Chords have at least four tones, which all belong to the simultaneous scale. The first and last chord root of a section is often the root of its scale. To ensure smooth transitions between chords, the voice-leading distance between consecutive chords is low (at most 3 semitones in the excerpt). The voice-leading distance is the minimal sum of absolute intervals between the tones of two chords. For example, the voice-leading distance between the C and A♭ major triads is 2 (C $\rightarrow$ C $=$ 0, E $\rightarrow$ E♭ $=$ 1, G $\rightarrow$ A♭ $=$ 1). Also, any three consecutive chords must be distinct.

The actual notes (in staves 1-3) must express the underlying harmony (stave 4). Nonharmonic tones (marked with an x in Fig. 3) are prepared and resolved by small intervals. Across the section starting in measure 8, the contrapuntal lines in the swell division rise gradually (pitch domain boundaries are rising), and melodic intervals are getting smaller (this section lasts over 10 bars, so this is not obvious from the few bars shown). The contrapuntal voices are never more than an octave apart; they don't cross; they avoid open and hidden parallels; they avoid perfect consonances between simultaneous notes (one is there in Fig. 3 after manual revisions); and voice notes sound all tones of the underlying harmony. Also, the lines are composed from motifs; and durational accents are constrained [20].

4. THE FRAMEWORK

This section describes formal details of the proposed framework. It explains how musical objects (notes, chords, and scales) are represented. Notes are concrete musical objects that produce sound when the score is played, but chord and scale objects represent analytical information – the underlying harmony.

Notes, chords and scales are represented by tuples of decision variables. When users define harmony models with this framework, they employ these objects and apply constraints between their variables. However, some wellformedness constraints must always hold between these variables, and these constraints are discussed here as well.

For clarity and portability, this section shows core definitions of the framework in mathematical notation instead of using any programming language (e.g., Oz). For simplicity, we leave out some auxiliary variables (intermediate results represented by extra variables).

4.1 Declaration of Chord and Scale Types

In the proposed framework, the chord and scale types supported globally by a constraint model (e.g., major and minor triads and scales) can be declared independently of the rest of the model. The ordered sequence $\mathcal{CT}$ consists of tuples, where each tuple specifies one chord type with a set of features as shown in the example below (1). The first tuple declares the major triad type: it specifies the pitch class integer representing the untransposed chord root (C), and the

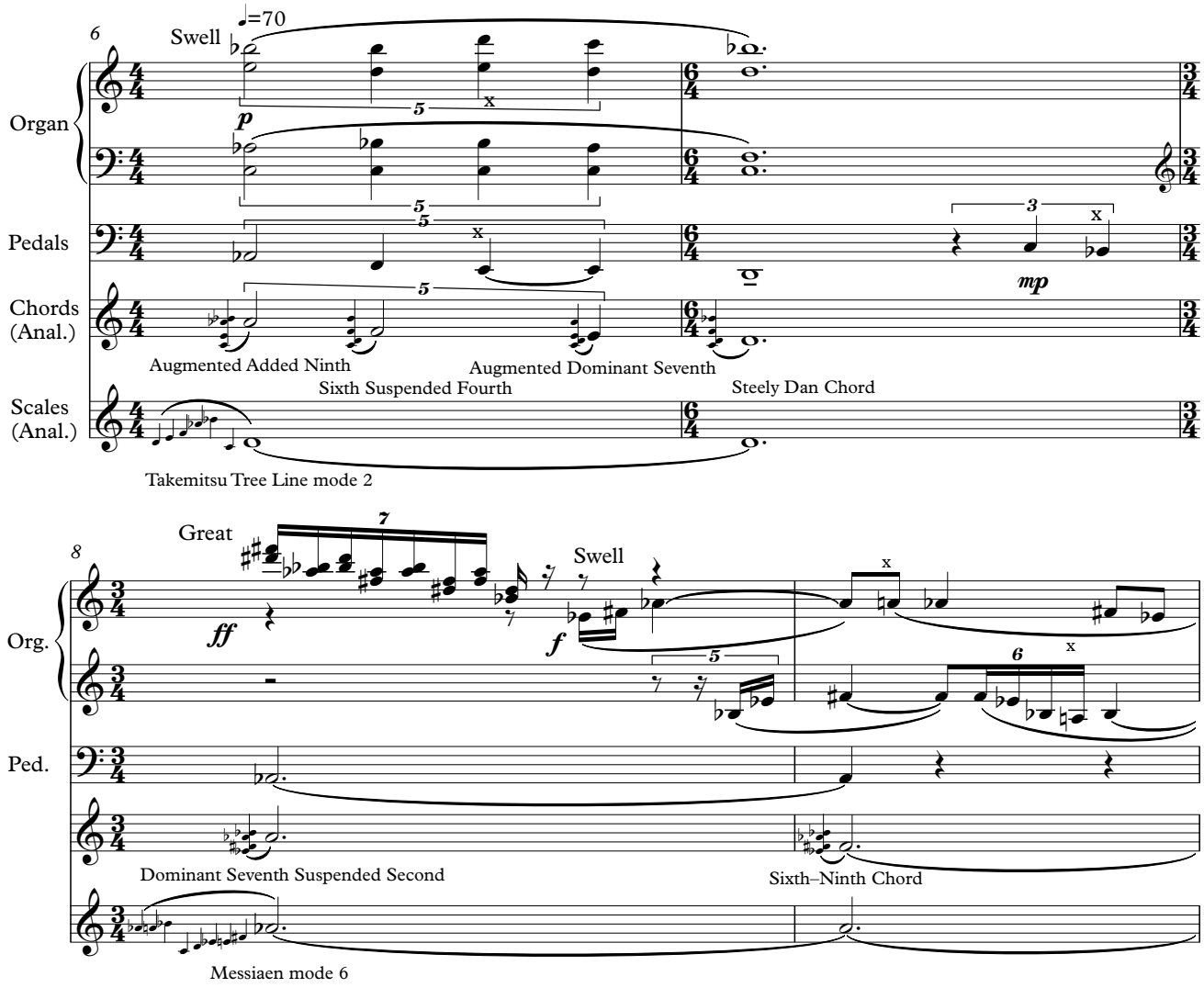

Figure 3. Excerpt from the composition *Pfeifenspiel* composed by the author. The upper three staves show the actual composition, and the lower two an analysis of the underlying harmony. Chord and scale tones are shown like an appoggiatura and roots as normal notes. Nonharmonic tones are marked by an x.

pitch classes of the untransposed chord – in this case the C-major triad, {C, E, G} – as a set of pitch class integers. The given $name$ is a useful annotation, but not directly used by the framework.

$$\mathcal{CT} = \begin{array}{l}[\langle name\text{: major},\ root\text{: } 0,\ PCs\text{: } \{0,4,7\}\rangle,\\ \ \langle name\text{: minor},\ root\text{: } 0,\ PCs\text{: } \{0,3,7\}\rangle,\\ \ \ldots]\end{array} \quad (1)$$

Scale types are declared in the same way in an extra sequence of tuples $\mathcal{ST}$. For example, the pitch class set of the major scale type is $\{0,2,4,5,7,9,11\}$, while its root is 0.

Users can also declare the global number of pitches per octave ($psPerOct$), and that way specify the meaning of all integers representing pitches and pitch classes in the constraint model. [4] A useful default value for $psPerOct$ is 12, which results in the common 12-EDO, and which was used for the chord and scale type examples above.

Instead of specifying pitch classes by integers as shown, it can be more convenient to specify note names, which are then automatically mapped to the corresponding pitch class integers, depending on $psPerOct$. In 12-EDO, C $\mapsto$ 0, C♯ $\mapsto$ 1 and so on. Alternatively, pitch classes can be specified by frequency ratios as a useful approximation of just intonation intervals for different temperaments. Again in 12-EDO, the prime $\frac{1}{1} \mapsto 0$, the fifth $\frac{3}{2} \mapsto 7$ etc. [5]

The format of chord and scale declarations is extendable. Users can add further chord or scale type features (e.g., a measure of the dissonance degree of each chord type), which would then result in further variables in the chord and scale representation discussed in the Sec. 4.3 below.

Note that chord and scale declarations are internally rearranged for the constraints discussed in Sec. 4.3. For example, $root_{\mathcal{CT}}$ is the sequence of all chord roots (in the order of the chord declarations), $PCs_{\mathcal{ST}}$ is the sequence of the pitch class sets of all scale declarations, and so on.

[4] Only equal temperaments that evenly subdivide the octave are supported. However, just intonation or irregular temperaments can be closely approximated by setting $psPerOct$ to a high value (e.g., $psPerOct = 1200$ results in cent resolution).

[5] Remember that for the frequency ratio r, the corresponding pitch class is $round((\log_2 r) \cdot psPerOct)$.

4.2 Temporal Music Representation

The underlying harmony can change over time. Temporal relations are a suitable way to express dependencies: all notes simultaneous to a certain chord or scale depend on that object (i.e., those notes fall into their harmony).

The framework shows its full potential when combined with a music representation where multiple events can happen simultaneously. A chord sequence (or scale sequence or both) can run in parallel to the actual score, as shown in the score example discussed above (Fig. 3).

Score objects are organised in time by hierarchic nesting. A sequential container implicitly constrains its contained objects (e.g., notes, chords, or other containers) to follow each other in time. The objects in a simultaneous container start at the same time (by default). All temporal score objects represent temporal parameters like their *start* time, *end* time and *duration* by integer variables. A rest before a score object is represented by its temporal parameter *offset* time (another integer variable), which allows for arbitrary rests between objects in a sequencial container, and before objects in a simultaneous container.

Equation (2) shows the constraints between temporal variables of a simultaneous container sim and its contained objects $\mathsf{object}_1 \ldots \mathsf{object}_n$. Any contained object – object_i – starts at the start time of the container sim plus the offset time of the contained object. The end time of sim is the maximum end time of any container. The relations between temporal variables of a sequential container and its contained objects are constrained correspondingly, and every temporal object is constrained by the obvious relation that the sum of its start time and duration is its end time. The interested reader is referred to [21, Chap. 5] for further details.

$$\begin{aligned} start_{\mathsf{object}_i} &= start_{\mathsf{sim}} + offset_{\mathsf{object}_i} \\ end_{\mathsf{sim}} &= \max(end_{\mathsf{object}_1}, \ldots, end_{\mathsf{object}_n}) \end{aligned} \tag{2}$$

Temporal relations can be defined with these temporal parameters. For example, we can constrain that (or whether, by using a resulting truth value) two objects o_1 and o_2 are simultaneous by constraining their start and end times (3). Note that for clarity this constraint is simplified here by leaving out the offset times of these objects, and remember that $\wedge$ denotes a conjunction (logical and).

$$start_{\mathsf{o}_1} < end_{\mathsf{o}_2} \wedge start_{\mathsf{o}_2} < end_{\mathsf{o}_1} \tag{3}$$

Remember that all these relations are constraints – relations that work either way. The temporal structure of a score can be unknown in the definition of a harmonic CSP. Users can apply constraints, e.g., to control the harmonic rhythm in their model, or the rhythm of the notes in a harmonic counterpoint.

If other constraints depend on which objects are simultaneous to each other (e.g., harmonic relations between notes and chords), then the search should find temporal parameters relatively early during the search process.

4.3 Chords and Scales

The proposed model represents the underlying harmony of music with chord and scale objects. This section introduces the representation of these objects, their variables, and the implicit constraints between these variables. The representation of chords and scales is identical, except that chords depend on the declaration of chord types $\mathcal{CT}$, and scales on scale types $\mathcal{ST}$ (see Sec. 4.1). Therefore, the rest of this subsection only discusses the definition of chords.

A chord c is represented by a tuple of four variables (4) – in addition to the temporal variables mentioned above (Sec. 4.2) that are indicated with "…". [6]

$$\mathsf{c} = \langle type,\ transp,\ PCs,\ root, \ldots \rangle \tag{4}$$

The *type* (integer variable) denotes the chord type. Formally, it is the position of the respective chord in the collection of chord type declarations $\mathcal{CT}$, see Eq. (1) above. The *transp* (integer variable) specifies how much the chord is transposed with respect to its declaration. *PCs* (set of integers variable) is the set of (transposed) pitch classes of the chord, and the *root* (integer variable) is the (transposed) root pitch class.

For chords where the *root* is 0 (C) in the declaration, *transp* and *root* are equal. In a simplified framework, the variable *transp* could therefore be left out. However, sometimes it is more convenient to declare a chord where the root is not C (e.g., leaving a complex chord from the literature untransposed, or stating the pitch classes of a chord by fractions where the root is not $\frac{1}{1}$). Therefore this flexibility is retained here with the separate variables *transp* and *root*.

The constrains (5) and (6) restrict the relation between the variables of any chord object c, and the collection of chord type declarations $\mathcal{CT}$. The element constraint is a key here. It accesses in an ordered sequence of variables a variable at a specific index, but the index is also a variable. [7] In (5), $root_{\mathcal{CT}}[type]$ is the untransposed root of the chord *type*. The (transposed) chord *root* is that untransposed root – pitch-class transposed by the constraint *transp-pc*. Pitch class transposition in 12-EDO with modulus 12 is well known in the literature. The definition here uses *psPerOct* as divisor to support arbitrary equal temperaments. A corresponding constraint for pitch class sets is expressed in (6).

$$root_{\mathsf{c}} = transp\text{-}pc(root_{\mathcal{CT}}[type_{\mathsf{c}}], transp_{\mathsf{c}}) \tag{5}$$

$$PCs_{\mathsf{c}} = transp\text{-}PCs(PCs_{\mathcal{CT}}[type_{\mathsf{c}}], transp_{\mathsf{c}}) \tag{6}$$

When chords are extended by further variables (e.g., a chord type specific dissonance degree) the chord declarations and chord object variables are simply linked by further element constraints (e.g., $feat_{\mathsf{c}} = feat_{\mathcal{CT}}[type_{\mathsf{c}}]$).

4.4 Notes with Analytical Information

Note objects represent the actual notes in the score. A note n is represented by a tuple of variables as shown in (7). As with chords, temporal variables are left out for simplicity, and are only indicated with "…".

[6] Internally, some additional auxiliary variables are used in the implementation, $untransposedRoot$ and $untransposedPitchClasses$.

[7] The element constraint is notated here like accessing an element in an array. $x = xs[i]$ constrains x to the element at position i in xs, where both x and i are integer or set variables, and xs is a sequence of such variables.

$$\mathsf{n} = \langle pitch,\ pc,\ oct,\ inChord?,\ inScale?, \dots \rangle \quad (7)$$

The note's $pitch$ (integer variable) is essential for melodic constraints. It is a MIDI note number in case of 12-EDO. The pc (integer variable) represents the pitch class (chroma) independent of the oct (octave, integer variable) component, which is useful for harmonic constraints. The relation between a note's $pitch$, pc and oct is described by (8); the octave above middle C is 4.

$$pitch = pc + (oct + 1) \cdot psPerOct \quad (8)$$

The Boolean variable $inChord?$ indicates a harmonic or nonharmonic note, i.e., whether the pc of a note n is an element of the PCs of its simultaneous chord c, implemented with a reified set membership constraint (9). The Boolean variable $inScale?$ denotes equivalently whether notes are inside or outside their simultaneous scale.

$$inChord?_{\mathsf{n}} = pc_{\mathsf{n}} \in PCs_{\mathsf{c}} \quad (9)$$

4.5 Degrees, Accidentals, and Enharmonic Spelling

So far, the formal presentation used two common pitch representations: the single variable $pitch$ and the pair $\langle pc, oct \rangle$; further pitch-related representations can be useful to denote scale degrees (including deviations from the scale), tones in a chord (and deviations), and to express enharmonic notation. These representations are closely related. They "split" the pc component into further representations, depending on a given scale or chord. These representations are only briefly summarised here; formal details are left out due to space limitations.

Enharmonic spelling is represented with the pair of integer variables $\langle nominal, accidental \rangle$, where $nominal$ represents one of the seven pitch nominals (C, D, E, ...) as integers: 1 means C, 2 means D, ..., and 7 means B. [8] The variable $accidental$ is an integer where 0 means ♮, and 1 means raising by the smallest step of the current equal temperament. In 12-EDO, 1 means ♯, -1 means ♭, and -2 is 𝄫 and so on. [9]

The representation of enharmonic spelling depends on the pitch classes of the C major scale: the $nominal$ 1 is a reference to the pitch class at the first scale degree of C-major, 2 refers to the second degree and so on. The same scheme can be used with any other scale to express scale degrees with the pair of integer variables $\langle scaleDegree, scaleAccidental \rangle$. The variable $scaleDegree$ denotes basically the position in the pitch classes of a given scale. If the variable $scaleAccidental$ is 0 (♮), then the expressed pitch class is part of that scale. Otherwise, $scaleAccidental$ denotes how far the expressed pitch class deviates from the pitch class at $scaleDegree$. This representation has been used in the Schoenberg example described in Sec. 3.2 to constrain the raised scale degrees I, II, IV, and V.

[8] The choice to start with C and not A as 1 is arbitrary, and it is very easy to change that when desired. C is represented by 1 and not 0 for consistency with scale degrees, where the lowest degree is commonly notated as I.

[9] This representation can also be implemented in a constraint system that only supports positive integer variables by adding a constant offset to all accidental variables.

This scheme can further be used for chords with the pair of integer variables $\langle chordDegree, chordAccidental \rangle$. The integer variable $chordDegree$ denotes a specific tone of a chord, e.g., its root, third, fifth etc.; it is the position of a chord tone in its chord type declaration in $\mathcal{CT}$, while $chordAccidental$ (integer variable) indicates whether and how much the note deviates from a chord tone (for nonharmonic tones). This representation was also used in the model of Schoenbergs theory of harmony discussed above to recognise dissonances that should be resolved (e.g., any seventh in a seventh chord).

5. MODELLING WITH THE FRAMEWORK

The proposed framework consists primarily of the constrainable harmony representation presented in the previous section. Developing concrete harmony models with this foundation is relatively straightforward: the variables in the representation are constrained. This section shows formal details of the first application shown in section 3.1.

For simplicity, this example only uses three chord types: major, minor, and the dominant seventh chord. These are declared exactly as shown previously in equation (1), where only the dominant seventh chord is added with the pitch classes $\{0, 4, 7, 10\}$ and root 0. The example declares scale types in the same way; only the major scale is needed.

The music representation of this example consists of a nested data structure. The top level is a simultaneous container, which contains three objects: a sequential container of notes; a sequential container of chords; and a scale. In the definition, the note pitches and temporal values are set to the melody (the folksong "Horch was kommt von draussen 'rein"), the scale is set to C major, and all chord durations are set to whole note values. In other words, only the chord types and transpositions, and hence also their pitch classes and roots are unknown in the definition.

Space does not permit to discuss all constraints of this example, but the formalisation of some of them gives an idea of the overall approach. As discussed before, the harmony of notes are their simultaneous chords and scale; chords also depend on their simultaneous scale. Only diatonic chords are allowed: every chord pitch class set is a subset of the scale pitch class set.

However, the example allows for nonharmonic notes. Constraining the notes' parameter $inChord?$ allows to model these. Nonharmonic notes are surrounded by harmonic notes, i.e., the parameter $inChord?$ of their predecessor and successor notes must be true. Only passing tones and neighbour tones are permitted: the melodic intervals between a nonharmonic note and its predecessor and successor notes must not exceed a step (two semitones). Equation (10) shows these constraints; remember that the operator $\Rightarrow$ indicates the implication constraint (logical consequence).

$$\begin{aligned} & inChord?_{\mathsf{n}_i} = \text{false} \Rightarrow \\ & \quad inChord?_{\mathsf{n}_{i-1}} = inChord?_{\mathsf{n}_{i+1}} = \text{true} \\ & \wedge\ |pitch_{\mathsf{n}_i} - pitch_{\mathsf{n}_{i-1}}| \leq 2 \\ & \wedge\ |pitch_{\mathsf{n}_{i+1}} - pitch_{\mathsf{n}_i}| \leq 2 \end{aligned} \quad (10)$$

The complete constraint problem definition is then given to the solver, which returns one or more solutions.

6. DISCUSSION

The applications shown in this paper demonstrate that the proposed framework is rather flexible. Complex theories of harmony in different styles can be modelled; in composition applications, generated notes can depend on an underlying harmony by ensuring a suitable treatment of nonharmonic tones; and in an analysis, generated chords can depend on given notes.

However, the proposed design is best suited for tonal music. For example, any atonal pitch class set can be declared as well, but then information like the chord root is redundant (it can simply be ignored in a model, though).

While the framework also supports tonal music in an extended sense (see Sec. 3.3) and microtonal music, it is less suitable for spectral music composition. Spectral music is based on absolute frequencies (and their intervals) translated into pitches. This approach preserves the octave of each pitch and that way the order of pitches in a chord. By contrast, in the proposed model chord and scale types are expressed by pitch classes. Individual chord or scale pitches can thus be freely octave transposed while retaining the chord or scale identity. Such an approach allows to control melodic and harmonic aspects independently with constraints.

The proposed model could be changed to better support spectral music by expressing chords with absolute pitches instead of pitch classes, and by disregarding all information based on pitch classes (chord roots, scale degrees etc.), but then tonal music theories depending on such analytical information that is independent of an octave component cannot be modelled anymore. The music constraint system PWMC [13] and its successor Cluster Engine implement such an approach.

A compromise could be special rules that constrain specific chord tones – e.g., tones at or above a certain $chord\text{-}Degree$ – into or above certain octaves, or above other chord tones, like some popular music voicing recommendations do (e.g., in a $V^{7\sharp 9}$ chord, the augmented ninth is preferred above the major third).

The framework supports microtonal music, but only equal divisions of the octave. Specifically, just intonation intervals are best represented by ratios, and unequal temperaments with floats, but the proposed framework only uses integers, because constraint propagation works very efficiently for those. Nevertheless, just intonation intervals can be closely approximated (see footnote 4).

In summary, this paper presents a framework for modelling harmony with constraint programming that is suitable for modern propagation-based constraint solvers. Its representation of harmonic concepts such as chords, scales, and a variety of pitch representations lead to harmony CSPs with a high level of abstraction. Three applications demonstrated the range and complexity of harmonic CSPs that are made possible.

7. REFERENCES

[1] A. Schoenberg, *Theory of Harmony*. University of California Press, 1983.

[2] M. Levine, *The Jazz Theory Book*. Sher Music Co., 1995.

[3] V. Persichetti, *Twentieth-Century Harmony: Creative Aspects and Practice*. W. W. Norton & Company, 1961.

[4] D. B. Doty, *The Just Intonation Primer. An Introduction to the Theory and Practice of Just Intonation*, 3rd ed. San Francisco, CA: Just Intonation Network, 2002.

[5] P. v. Roy and S. Haridi, *Concepts, Techniques, and Models of Computer Programming*. Cambridge, MA: MIT Press, 2004.

[6] K. R. Apt, *Principles of Constraint Programming*. Cambridge: Cambridge University Press, 2003.

[7] F. Pachet and P. Roy, "Musical Harmonization with Constraints: A Survey," *Constraints Journal*, vol. 6, no. 1, pp. 7–19, 2001.

[8] T. Anders and E. R. Miranda, "Constraint Programming Systems for Modeling Music Theories and Composition," *ACM Computing Surveys*, vol. 43, no. 4, pp. 30:1–30:38, 2011.

[9] T. Anders, "Compositions Created with Constraint Programming," in *The Oxford Handbook of Algorithmic Music*, A. McLean and R. T. Dean, Eds. Oxford University Press, forthcoming.

[10] K. Ebcioğlu, "Report on the CHORAL Project: An Expert System for Harmonizing Four-Part Chorales," IBM, Thomas J. Watson Research Center, Tech. Rep. Report 12628, 1987.

[11] C. Rueda, M. Lindberg, M. Laurson, G. Block, and G. Assayag, "Integrating Constraint Programming in Visual Musical Composition Languages," in *ECAI 98 Workshop on Constraints for Artistic Applications*, Brighton, 1998.

[12] M. Laurson, "PATCHWORK: A Visual Programming Language and some Musical Applications," PhD thesis, Sibelius Academy, Helsinki, 1996.

[13] Ö. Sandred, "PWMC, a Constraint-Solving System for Generating Music Scores," *Computer Music Journal*, vol. 34, no. 2, pp. 8–24, 2010.

[14] F. Pachet, "An Object-Oriented Representation of Pitch-Classes, Intervals, Scales and Chords: The basic MusES," LAFORIA-IBP-CNRS, Universite Paris VI, Tech. Rep. 93/38, 1993, revised and extended version.

[15] P. Roy and F. Pachet, "Reifying Constraint Satisfaction in Smalltalk," *Journal of Object-Oriented Programming*, vol. 10, no. 4, pp. 43–51, 1997.

[16] F. Benhamou, N. Jussien, and B. O'Sullivan, Eds., *Trends in Constraint Programming*. London, UK: ISTE, 2013.

[17] T. Anders and E. R. Miranda, "A Computational Model that Generalises Schoenberg's Guidelines for Favourable Chord Progressions," in *6th Sound and Music Computing Conference*, Porto, Portugal, 2009, pp. 48–52.

[18] ——, "A Computational Model for Rule-Based Microtonal Music Theories and Composition," *Perspectives of New Music*, vol. 48, no. 2, pp. 47–77, 2010.

[19] D. Tymoczko, *A Geometry of Music: Harmony and Counterpoint in the Extended Common Practice*. Oxford: Oxford University Press, 2011.

[20] T. Anders, "Modelling Durational Accents for Computer-Aided Composition," in *Proceedings of the 9th Conference on Interdisciplinary Musicology CIM14*, Berlin, Germany, 2014, pp. 90–93.

[21] ——, "Composing Music by Composing Rules: Design and Usage of a Generic Music Constraint System," Ph.D. dissertation, School of Music & Sonic Arts, Queen's University Belfast, 2007.